In the Specification

Please delete paragraph [001] in its entirety and replace it with the following:

[001] This application is related to U.S. Patent Application Serial No. 10/696,968 [[_____]], entitled SYSTEM AND METHOD FOR COBOL TO PROVIDE SHARED MEMORY AND MEMORY AND MESSAGE QUEUES, inventor Joseph G. Laura, filed on even date herewith, U.S. Patent Application Serial No. 10/697,417 [[_____]], entitled SYSTEM AND METHOD FOR ASYNCHRONOUS PROCESSING IN COBOL, inventor Joseph G. Laura, filed on even date herewith, and U.S. Patent Application Serial No. 10/696,895 [[_____]], entitled IMPLEMENTATION OF DISTRIBUTED AND ASYNCHRONOUS PROCESSING IN COBOL, inventor Joseph G. Laura, filed on even date herewith, all of which are incorporated herein by reference for all purposes.

Please delete paragraph [008] in its entirety and replace it with the following:

[008] The processing techniques described above are examples of useful functionality widely available to programmers using distributed and asynchronous processing languages, such as C and JAVA, but unavailable in COBOL. Frequently, it is desirable for business processes employing COBOL applications to accomplish distributed and asynchronous processing. Although the COBOL language has limitations, it is difficult for businesses with a significant investment in COBOL programs to justify abandoning the COBOL applications and redeveloping the applications using a more modern and flexible language, such as C or JAVA. Instead, COBOL systems are typically provided with an interface or "hook" to enable the COBOL program to cooperate with, for example, C or JAVA programs. The C or JAVA [[Java]] program then performs the distributed and

asynchronous processing tasks that the COBOL application is otherwise incapable of handling independently.

Please delete paragraph [027] in its entirety and replace it with the following:

[027] As discussed above, distributed and asynchronous processing is intended to describe a variety of functionality that is not native to COBOL or was not previously available to in the COBOL programming language, but which is available in distributed and asynchronous processing environments and programming languages such as C and JAVA [[Java]]. The terms distributed and/or asynchronous processing, as used herein, are intended to include, but not be limited to, one or more of the programming techniques and functionality for programming, enabling, and managing sockets and pipes, shared memory, threads, memory and message queues, signal handlers, events, semaphores and mutexes.

Please delete paragraph [051] in its entirety and replace it with the following:

[051] Figure 4 is a block diagram illustrating the threads routine 20d of the technical layer 10. Threads provide for asynchronous processing by multiple processes simultaneously. The threads routine 20d achieves the functionality of native threads, as used in C and JAVA [[Java]], and enables them for the COBOL language by employing and managing subtasks, as described below, which were not previously used by COBOL programs in this manner. The threads routine 20d enables COBOL applications to be split into multiple paths. Figure 4 illustrates a first COBOL program 50 and a second COBOL program 52, which are both similar to the COBOL program 12 illustrated in

Figure 1. The threads routine 20d allows the first and second COBOL programs 50 and 52 to use a shared memory 54 to enable threads for COBOL. The operation and function of shared memory will be discussed in greater detail hereinafter with regard to Figure 7.

Please delete paragraph [055] in its entirety and replace it with the following:

[055] Employing the threads routine 20d, COBOL programmers can now break-up large database file reading operations into multiple threads to read through large files and databases much more rapidly and efficiently. Enabling this functionality required, among other things, identifying the differences between mainframe COBOL programming and threads as natively used in languages such as C or JAVA [[Java]]. For example, threads natively share, for example, address space, process, memory and file space as well as returning a message in the event a thread abruptly terminates. To enable threads for COBOL, the technical layer 10 uses subtasks. COBOL was not previously operable to support subtasks in this manner. Subtasks use the same address space, have distinct process IDs, and share memory, but not file space. As discussed above, subtasks do not provide notification when they abnormally terminate. The threads routine 20d must manage these and other aspects of subtasks. In the present embodiment, the threads routine 20d of the technical layer 10 uses, among other techniques, subtasks to accomplish the thread functionality that is natively supported in computer languages such as C or JAVA [[Java]].

Please delete paragraph [071] in its entirety and replace it with the following:

[071] The message queue functionality is enabled by the message queue routine 20h via a call to the operating system 34 [[32]], as discussed above. As is the case with the memory queue routine 20g, the message queue routine 20h includes a plurality of functions to manage message queues including a create function to create new queues such as by initiating an appropriate operating system call. Other functions include an attach function for connecting to existing queues and a query function to determine whether a queue exists and obtain the size of the queue in terms of the number of rows as well as additional information related to the queue. Some other functions of the message queue routine 20h include a push function to add a row to the queue and a block function to block when the queue is full. A pop function removes the top row from the queue and also prevents this operation when no more rows exist on the queue. Additional functions provide for detaching from an existing queue and removing the queue from the system.

Please delete paragraph [074] in its entirety and replace it with the following:

[074] Figure 7 illustrates one embodiment of the shared memory routine 26 for sharing memory between COBOL programs. The shared memory routine 20c is provided with an index 110 for maintaining a plurality of keys 112 related to an address 114 in memory 36 [[26]] that is used as shared memory by the first and second COBOL programs 50 and 52. The first COBOL program 50 requests shared memory from the shared memory routine 20c. The shared memory routine 20c includes a plurality of functions operable for managing shared memory, including a create function for creating a shared memory

block. In the present embodiment, the shared memory routine 20c is a COBOL program that issues a call to the operating system 34 to allocate the appropriate amount of memory.

Please delete paragraph [077] in its entirety and replace it with the following:

[077] As previously discussed, the linkage section 96a of the first COBOL program 50 is typically used for passing information between subprograms or calling programs, but is employed in a novel manner in the present embodiment to map to the address 114 of the memory 36 [[26]] that is used for shared memory. Mapping the address 114 to the linkage section 96a of the first COBOL program 50 is useful since shared memory only needs to be loaded one time and does not require constant refreshing. For example, data space is known as a section of mainframe memory that stays resident when programs exit, but requires a task running to keep the memory refreshed. Core loads are another example of memory employed in mainframe computer systems, but when the program using the memory exits or closes, the memory is released and the data is no longer available. Employing the shared memory routine 20c according to the present aspect, the memory is maintained by the shared memory routine 20c even after the first COBOL program 50 terminates.

Please delete paragraph [089] in its entirety and replace it with the following:

[089] When the COBOL program 12 initiates the child process 156, such as child process 156a, the COBOL program 12 puts the child process 156a into an event wait. In this embodiment, the events routine 20b may place the child process 156a into the event

wait. The child process 156a registers with the COBOL program 12 such that the ~~process~~ PID 152 and event that the child process 156a is waiting on are recorded in the index 150 of the COBOL program 12. The COBOL program 12 may obtain a ~~process~~ PID 152 and event 154 associated with the child process 156a on initializing or on starting the child process 156a or may obtain this information subsequently from the events routine 20b. In some cases, the child process 156a would not be required to register with the COBOL program 12.

Please delete paragraph [090] in its entirety and replace it with the following:

[090] In the present embodiment, the child process 156a registers with a register 158 of the events routine 20b. The register 158 maintains the ~~process~~ PID 152 and event 154 associated with the child process 156a. The events routine 20b maintains the functionality to place the child process 156a, as well as the other child processes 156b and 156c, into the wait state upon request by the COBOL program 12.

Please delete paragraph [091] in its entirety and replace it with the following:

[091] At the appropriate time, the COBOL program 12 initiates the child process 156a by signaling the events routine 20b using the ~~process~~ PID 152 and the event 154 associated with the desired process, such as the child process 156a. In some cases, the COBOL program 12 may only signal the child process 156a using the ~~process~~ PID 152, via the events routine 20b. The events routine 20b receives the ~~process~~ PID 152 from the COBOL program 12 and associates the event 154 with the ~~process~~ PID 152 using the

register 158. The events routine 20b then signals the appropriate child process, such as child process 156a, based on the ~~process~~ PID 152.

Please delete paragraph [092] in its entirety and replace it with the following:

[092] The child process 156a is programmed to perform a process, such as outputting to a shared resource 56 or writing to memory, for example. The COBOL program 12 and the events routine 20b, via the index 150 and register 158, respectively, are operable to maintain a plurality of ~~process~~ PIDs 152 and a plurality of events 154 associated with the one or more child processes 156. The events routine 20b is operable to coordinate signaling and processing of the child processes 156 to the shared resource 56.

Please delete paragraph [100] in its entirety and replace it with the following:

[100] The customer interface 180 may be comprised of a plurality of COBOL programs including a profile changes program 190. The profile changes program 190 is operable using the message queue routine 20h of the technical layer 10 to employ an input message queue 192. The profile changes program 190 using the shared memory routine 20c is operable to establish a shared memory 194 on the second machine 174. To quickly and efficiently read thousands of records from the customer master file, the profile changes program 190 is operable, using the technical layer 10, to spawn a plurality of jobs 196 designated alphanumerically 196a, 196b, and 196c. Processing the separate jobs 196 may be accomplished, for example, using the semaphore routine 20i, as described above. In this manner, the profile changes program 190 breaks up access to

the customer master file into multiple jobs to improve the efficiency of accessing the customer master information.

Please delete paragraph [101] in its entirety and replace it with the following:

[101] Each of the jobs 196 operates in its own address space, but is able to share the shared memory 194. As each of the jobs 196 is started, the jobs 196 look at the shared memory 194 to obtain the key with the location to the input message queue 192, such as elsewhere in memory. The jobs 196 then take the first message off the input message queue 192 and process the message or instruction. The job 196 then, for example, pulls or reads records off the customer master file and puts or writes the information to an output message queue 198. The profile changes program 190, via the technical layer 10, is operable to monitor and manage the input message queue 192 to provide the jobs 196 with messages or work. The shared memory 196 is used to communicate across the jobs 196 any information to be shared between the jobs 196.

Please delete paragraph [102] in its entirety and replace it with the following:

[102] An A/R changes program 200 is a COBOL program or routine that is operable to read information from the output message queue 198 as written from the jobs 196. The A/R changes program 200 may be a program or routine that is part of the customer interface 180. The A/R changes program 200, in this example, requires information from the first and second A/R files 182 and 184 to complete processing. Rather than route jobs one-at-a-time and waiting until the process is returned, the A/R changes program 200 is operable, using the technical layer 10, to do a socket connection request directly to

the relevant A/R customer data. Specifically, the A/R changes program 200, using the sockets routine 20e, is operable to enable socket communication to the first A/R file 182 on the first machine 172 and further operable to retrieve information from the second A/R file 184 on the third machine 176 via a socket request.

Please delete paragraph [103] in its entirety and replace it with the following:

[103] The A/R changes program 200 combines the information from the output message queue 198 received from the jobs 196 with the associated data from the first and second A/R files, received via the socket connections. The A/R changes program 200 then outputs the combined information or data to program 202 or system on the fourth machine 178, via an additional socket communication. The output to the fourth machine 178 may be performed using, among other functions, the semaphore routine 20i. The program 202 is operable to appropriately divide and store the data into the databases 186.